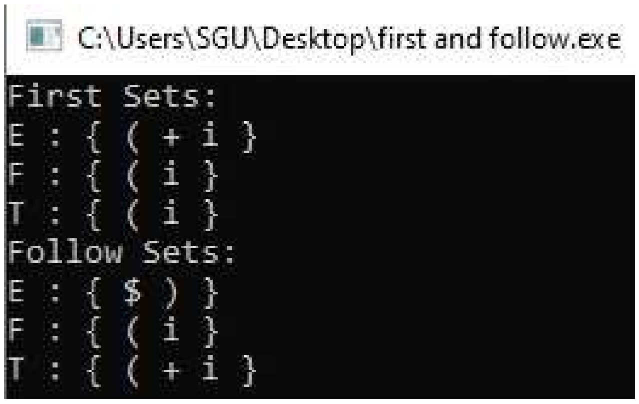
Name: Anushka Harshavadan Nevgi
Experiment: 5. Implement a program to apply first and follow functions of LL(1) class of grammar
Class: TY CSE A, 31

```
#include <iostream>
#include <vector>
#include <set>
#include <map>
#include <string>
#include <algorithm>

using namespace std;

class LL1Parser {
public:
    map<char, set<string>> productions;
    map<char, set<char>> firstSet;
    map<char, set<char>> followSet;
    set<char> nonTerminals;
    set<char> terminals;

    void addProduction(char nonTerminal, const
string& production) {
        productions[nonTerminal].insert(production);
        nonTerminals.insert(nonTerminal);
        for (char c : production) {
            if (isTerminal(c)) {
                terminals.insert(c);
            }
        }
    }

    void computeFirst() {
        bool changed;
        do {
            changed = false;
            for (auto& prod : productions) {
                char nonTerminal = prod.first;
                for (const string& production :
prod.second) {
                    for (char symbol : production) {
                        if (isTerminal(symbol)) {
                            if
(firstSet[nonTerminal].insert(symbol).second) {
                                changed = true;
                            }
                            break;
                        } else if
(nonTerminals.count(symbol)) {
                            for (char f : firstSet[symbol]) {
                                if
(firstSet[nonTerminal].insert(f).second) {
                                    changed = true;
                                }
                            }
                            if (firstSet[symbol].count('e') == 0)
{
                                break;
                            }
                        }
                    }
                }
            }
        } while (changed);
    }

    void computeFollow() {
        followSet['E'].insert('$');  // Add '$' to the
follow set of the start symbol 'E'
        bool changed;
        do {
            changed = false;
            for (auto& prod : productions) {
                char nonTerminal = prod.first;
                for (const string& production :
prod.second) {
                    for (size_t i = 0; i < production.size();
++i) {
                        if (isNonTerminal(production[i])) {
                            if (i + 1 < production.size()) {
                                if (isTerminal(production[i +
1])) {
                                    if
(followSet[production[i]].insert(production[i +
1]).second) {
                                        changed = true;
                                    }
                                } else {
                                    for (char f :
firstSet[production[i + 1]]) {
                                        if
(followSet[production[i]].insert(f).second) {
                                            changed = true;
                                        }
                                    }
                                    if (firstSet[production[i +
1]].count('e') == 0) {
                                        break;
                                    }
                                }
                        } else {
                            for (char f :
followSet[nonTerminal]) {
                                if
(followSet[production[i]].insert(f).second) {
                                    changed = true;
                                }
                            }
                        }
                    }
                }
            }
        }
    } while (changed);
}

void printSet(const map<char, set<char>>& sets)
{
    for (auto& s : sets) {
        cout << s.first << " : { ";
        for (auto& elem : s.second) {
            cout << elem << " ";
        }
        cout << "}" << endl;
    }
}

void printFirstSets() {
    cout << "First Sets:" << endl;
    printSet(firstSet);
}

void printFollowSets() {
    cout << "Follow Sets:" << endl;
    printSet(followSet);
}

private:
    bool isTerminal(char c) {
        return !(isNonTerminal(c) || c == 'e');
    }

    bool isNonTerminal(char c) {
        return nonTerminals.count(c);
    }
};
```

```
int main() {
    LL1Parser parser;
    parser.addProduction('E', "TE'");
    parser.addProduction('E', "+TE'");
    parser.addProduction('E', "e");
    parser.addProduction('T', "FT'");
    parser.addProduction('T', "e");
    parser.addProduction('F', "(E)");
    parser.addProduction('F', "id");

    parser.computeFirst();
    parser.computeFollow();

    parser.printFirstSets();
    parser.printFollowSets();
    return 0;
}
```



C:\Users\SGU\Desktop\first and follow.exe

```
First Sets:
E : { ( + i }
F : { ( i }
T : { ( i }
Follow Sets:
E : { $ ) }
F : { ( i }
T : { ( + i }
```